PATENT
5150-76801

"EXPRESS MAIL" MAILING
LABEL NUMBER: EV318247475US
DATE OF DEPOSIT: NOVEMBER 14, 2003
I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE "EXPRESS MAIL POST OFFICE TO ADDRESSEE" SERVICE UNDER 37 C.F.R. § 1.10 ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER OF PATENTS AND TRADEMARKS, ALEXANDRIA, VA 22313-1450

Derrick Brown

USER CONFIGURABLE DATA MESSAGES IN INDUSTRIAL NETWORKS

By:

Zaki Chasmawala
Brian Ruhmann
Sadia Malik

Attorney Docket No.: 5150-76801

Jeffrey C. Hood/MRS
Meyertons, Hood, Kivlin, Kowert & Goetzel PC
P.O. Box 398
Austin, Texas 78767-0398
Ph: (512) 853-8800

Priority Claim

This application claims benefit of priority of U.S. provisional application Serial No. 60/426,718 titled "User Configurable Data Messages in Industrial Networks" filed November 15, 2002, whose inventors were Zaki Chasmawala, Brian Ruhmann, and Sadia Malik.

Field of the Invention

The present invention relates to the field of communication networks. In particular, the invention relates to a system and method for user configurable data messages on a communication network.

Description of the Related Art

Figure 1 (Prior Art)

Figure 1 illustrates a general bus (network) 2 that may have a serial data connection and may be operable to connect field devices such as sensors, actuators, drives, distributed input and/or output (I/O), and/or distributed regulators, with the controlling personal computer (PC) or a Programmable Logic Controller (PLC) 1. A controller area network (CAN) is a serial, asynchronous, multimaster communication protocol for connecting electronic control modules in automotive and industrial applications. CAN was designed for applications needing high-level data integrity and data rates of up to 1 Mbit/s.

Historically, CAN networks were predominantly automotive. Because of its technical merits and low cost, CAN has found acceptance in markets other than automotive. Some examples are aerospace, maritime, medical equipment, farm machinery, manufacturing (motion control, robots) etc. CAN is now being used not only in network automotive devices (sensors and actuators), but also in medical equipment (x-ray machines), robots, copy machines, and recreation vehicles, among others.

Using CAN, devices (controllers, sensors, and actuators) 3-8 are connected on a common serial bus 2. This network of devices can be thought of as a scaled-down and real-time

version of networks used to connect personal computers. Any device on a CAN network 2 can communicate with any other device using a common pair of wires.

On a CAN network 2, every CAN message may be given a unique arbitration ID. The arbitration ID determines the device's priority to transmit CAN messages on the CAN network. The lower the number of the arbitration ID, the higher is its priority. The arbitration ID may be determined by the system designer at network design time. Devices that perform critical functions are given higher priority arbitration IDs. Because of this arbitration scheme, no two devices with the same arbitration ID can exist on a CAN network.

Figure 2 (Prior Art)

A CAN message may contain eight bytes in which information can transmitted, as illustrated in Figure 2. A CAN device can be considered a smart sensor, or an actuator, that multiplexes measurement data from traditional A/D or D/A converters or discrete bits (on/off) in the eight data bytes. It may support more than one CAN message depending on the design of the device, the resolution of the digitized data and the number of interfaced sensors or actuators. The CAN device may then transmit the resulting CAN message with input (measurement) or sensor data or accept a CAN message from the network to update the output or actuator.

Figure 3 (Prior Art)

A CAN device may be designed for a specific application, as illustrated in Figure 3, in which it may service a limited number of inputs and/or outputs and utilize a fixed number of CAN messages in which the data is pre-allocated in the data bytes of the CAN message.

Network modules currently exist with associated input and output channels (I/O channels) that are operable to expose the I/O channels packed in CAN messages. Essentially, these network modules and the associated I/O channels become CAN devices on the network. They may allow custom configuration of the I/O channels, such as

sensor data, in CAN data messages. For example, for input I/O, the network module transmits data messages on the CAN network. The network module also accepts messages from the CAN network to update output I/O channels.

Several applications demand flexibility in the configuration of the data message, such as the CAN data message, and the way the data is transmitted on the network. It may become necessary to configure a data message that groups measurements from a certain physical location. Another application may require measurements to be grouped together to be transmitted in a high priority periodic data message. Some applications may also need to transmit a data message that contains analog data on a digital trigger. A similar application may need to transmit a data message with analog data when one of the data values crosses a threshold level or a percent dead band.

Summary of the Invention

One embodiment of the invention comprises a communication network that includes network devices. The network devices can communicate with each other using the communication network by transmitting and receiving data messages. A first network device and a second network device may each include one or more inputs and one or more outputs. The second network device may be coupled to a host computer. A data message may be created and sent by the first network device to the second network device, where the first data message includes user configurable data, where the user configurable data is configured using the host computer. A network device may offer the flexibility to allocate both analog and discrete data to specific data bytes in the same data message.

A network device, such as a CAN device, may be designed in the form of a network device that can be connected to different I/O interfaces, for signal conditioning and digitizing to various sensor types. For example, this network device may have a default mechanism of configuring the digitized sensor information in CAN messages on initialization. This network device may assign I/O data to the data messages and may provide the capability for a user to specify the location in the data bytes of the data message where the sensor information should be inserted.

The network device may also provide information about how much data space remains in the data message, such as a CAN data message, as well as provide information about the signals that have not been assigned to a data message. The network device may communicate with a graphical configuration tool on a host computer that will graphically display the data message configuration and allow a system designer to manipulate the data bytes. The network device may also permit the allocation of a new data message identifier to accommodate existing network devices.

Brief Description of the Drawings

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

Figure 1 illustrates a Prior Art topology of a general industrial network that includes a control computer and distributed I/O and/or sensors/actuators;

Figure 2 illustrates Prior Art various frame formats of data messages;

Figure 3 illustrates a Prior Art device with fixed frame formats;

Figure 4 is a block diagram of a network and a reconfigurable network device, according to one embodiment;

Figure 5 is a block diagram of a reconfigurable network device, according to one embodiment;

Figure 6 is a table of an exemplary channel and arbitration ID scheme for initialization, according to one embodiment;

Figure 7 is a table of an exemplary channel and arbitration ID scheme after a modification where all channels are used, according to one embodiment;

Figure 8 is a graphical view of an exemplary arbitration ID, according to one embodiment;

Figure 9 illustrates an application used to configure a network, according to one embodiment;

Figure 10 illustrates scanning and listing devices on the network, according to one embodiment;

Figure 11 illustrates configuration of a network device on the industrial network, including I/O channels, according to one embodiment;

Figure 12 illustrates configuration of an I/O point of the network device, according to one embodiment;

Figure 13 illustrates configuration of a data message, according to one embodiment;

Figure 14 illustrates saving of the configuration of the network device, according to one embodiment;

Figure 15 illustrates a graphical configuration tool accessing channels on the network, according to one embodiment;

Figure 16 illustrates the graphical configuration tool accessing channels on the network, according to one embodiment;

Figure 17 illustrates the graphical configuration tool accessing the network device, according to one embodiment;

Figure 18 illustrates the graphical configuration tool monitoring data on the network, according to one embodiment;

Figure 19 illustrates accessing a test panel to test channels, according to one embodiment;

Figure 20 illustrates using the test panel to read and write values, according to one embodiment;

Figure 21 illustrates accessing channel properties page, according to one embodiment;

Figure 22 illustrates a diagram of a graphical program that accesses data on the network, according to one embodiment; and

Figure 23 illustrates a front panel of the graphical program that accesses data on the network, according to one embodiment.

Description

Figure 4 - Block Diagram of a Network and a Reconfigurable Network Device

Figure 4 illustrates a block diagram of a network and a reconfigurable network device 10, such as a reconfigurable CAN device, according to one embodiment. The network device 10 is an interface that communicates via a local bus 15 with the I/O interfaces 12-14 that allow connections to sensors 18 and/or actuators 19. The network device 10 also may communicate with other devices 5 – 6 on the network 30. Furthermore, in one embodiment, software, such as described in above with reference to Figures 9 – 23, may consist of configuration software, such as a graphical configuration tool, running on a host computer 1 that is capable of transmitting commands via messages to the network device 10. A network interface, such as a second network device coupled to the host computer 1, may be necessary for the host computer 1 to connect to the network 30.

In one embodiment, the network device 10 may consist of a microprocessor (not shown), non-volatile memory (not shown), a network controller (not shown), such as a CAN network controller with a physical layer interface, and a local bus interface (not shown) to communicate with the I/O interfaces. The microprocessor may execute firmware that detects and configures the I/O interfaces, determines data size of an associated channel and allocates the channel to a data message, such as a CAN data message. The firmware may also accept configuration commands from the host computer via the CAN interface and perform runtime CAN functions for the network device. The configuration may be saved to the non-volatile memory of the network device.

In one embodiment, the network device 10 may allocate a range of 255 arbitration IDs. The network device 10 may allocate arbitration IDs required for accommodating the I/O channels from the connected I/O interfaces 12-14. The network device may reserve the first two arbitration IDs beginning with the base arbitration ID for configuration requests and responses. The remaining 253 arbitration IDs may be used for I/O channels. In one embodiment, the first ID, also referred to as the base arbitration ID, can be set either by

DIP switches on the network device 10 or by the user from the host computer 1, among others.

It is noted that the block diagram of Figure 4 is exemplary only. Further, various blocks of Figure 4 may be present in different order than that shown, or may not be present, as desired. Also, various additional blocks may be included as desired.

Figure 5 – Block Diagram of a Reconfigurable Network Device

Figure 5 illustrates a block diagram of a reconfigurable network device, according to one embodiment.

In one embodiment, on initial power-up of the network device, the network device 10 may detect the base arbitration ID, such as from the DIP switches or non-volatile memory. The network device may consecutively allocate arbitration IDs, as offset from the base arbitration ID, and allocate channels to data bytes beginning with the first I/O (input or output) interface. On completely populating the 8 data bytes of the data message, such as the CAN data message, the network device may allocate another data message if it detects unallocated channels, and begin the allocation process again. If an I/O interface consists of both input and output I/O types, then the network device may allocate a new arbitration ID after populating an arbitration ID with one type of I/O.

In one embodiment, Figure 5 illustrates a block diagram of a reconfigurable network device, such as a reconfigurable CAN device. For example, a CAN network interface 11 may have an base arbitration ID of 0x300. The CAN network interface 11 may interface to an analog input I/O interface 12, an analog output I/O interface 13, and a discrete input I/O interface 14, all connected by a local bus 15.

It is noted that the block diagram of Figure 5 is exemplary only. Further, various blocks of Figure 5 may be present in different order than that shown, or may not be present, as desired. Also, various additional blocks may be included as desired.

Figure 6 – Table of Channel and Arbitration ID Scheme on Initialization

Figure 6 illustrates an exemplary initial channel and arbitration ID scheme for initialization, according to one embodiment. For example, the arbitration IDs 0x300 and 0x301 may be reserved for configuration response and request. Arbitration IDs 0x302 and 0x303 may be reserved for analog input channels. Arbitration IDs 0x304 and 0x305 may be reserved for analog output channels. Arbitration IDs 0x306 and 0x307 may be reserved for discrete input and discrete output channels respectively.

In one embodiment, once the configuration step is completed the network device may be switched to a run mode, where it may transmit messages, such as CAN messages, that have input I/O channels, and accepts messages, such as CAN messages, from the network to update any outputs.

It is noted that the table of Figure 6 is exemplary only. Further, various entries of Figure 6 may be present in different order than that shown, or may not be present, as desired. Also, various additional entries may be included as desired.

Figure 7 – Table of Channel and Arbitration Scheme after Modification

Figure 7 illustrates example channel and arbitration ID scheme after a modification where all channels are used, according to one embodiment.

In one embodiment, a system designer may choose which channel is included in a data message, such as a CAN data message, using a graphical configuration tool, such as described below with reference to Figures 9-11. The network device 10 may allocate analog and discrete channels of the same type (either input or output), in the same data message. A new analog or discrete channel may be inserted at a new byte boundary within the 8 data bytes. A discrete channel from one I/O interface can be allocated adjacent to a channel from another I/O interface within the same byte. The hardware interface may allow empty bit spaces within a byte, but may not allow an entire empty byte space except at the end of the data message. On every user interaction, the graphical configuration tool on the host computer may be updated with the network device

configuration. Table in Figure 7 shows an exemplary configuration of I/O channels and reallocated IDs, and table in Figure 8 shows a graphical view of an exemplary arbitration ID (0x305).

For example, the arbitration IDs 0x300 and 0x301 may be reserved for configuration response and request. Arbitration ID 0x305 may be reserved for two analog input channels and eight discrete input channels. Arbitration ID 0x30A may be reserved for three analog input channels and eight discrete input channels. Arbitration ID 0x310 may be reserved for one analog output channel and sixteen discrete output channels. Arbitration ID 0x312 may be reserved for 3 analog output channels. Arbitration ID 0x31F may be reserved for four analog output channels. Arbitration ID 0x320 may be reserved for three analog input channels. In one embodiment, each analog input or output channel uses two bytes. In one embodiment, each discrete input or output channel uses one bit.

It is noted that the table of Figure 7 is exemplary only. Further, various entries of Figure 7 may be present in different order than that shown, or may not be present, as desired. Also, various additional entries may be included as desired.

Figure 8 – Graphical View of Exemplary Arbitration ID

Figure 8 illustrates a graphical view of an exemplary arbitration ID (0x305), according to one embodiment.

In one embodiment, the exemplary arbitration ID, 0x305, may contain two analog input channels and eight discrete input channels. For example, bytes 0 and 1 may contain data for analog input channel 1, two bytes per analog channel. Bytes 2 and 3 may contain data for analog input channel 2, two bytes per analog channel. Byte 4 may contain data for discrete input channels 1-8, one bit per discrete channel.

It is noted that the table of Figure 8 is exemplary only. Further, various entries of Figure 8 may be present in different order than that shown, or may not be present, as desired. Also, various additional entries may be included as desired.

Figures 9-11 – Graphical Configuration of I/O Channels on Network Device

According to one embodiment, Figures 9-11 illustrate graphical configuration of I/O channels on the network device 10 using a graphical configuration tool. After the I/O channels and CAN messages are configured, the configuration can be loaded into the graphical configuration tool and accessed from another application, such as a graphical program. The graphical configuration tool may expose the configuration parameters for the network device to an API that is accessible by programming development environments, such as a graphical program development environment, or a textual program development environment, such as a Microsoft Studio, Java, and C/C++, among others. As a result, offset and scaling information to decode the I/O data may not be necessarily contained in a data message.

In one embodiment, the graphical configuration tool can be used to configure data messages to be transmitted in several different ways:

periodical determinism,
change of a state,
reaching a predetermined level, and
poll from the communication network, among others.

For example, users may benefit from this feature by having data, such as temperature data, assigned to messages that are transmitted only on change of state, and have other analog or discrete data periodically transmitted. The level based transmission is particularly useful for alarming purposes. A level based message can be transmitted either periodically or once on reaching a high or low level. The network device may allow full configuration of channels, including the way they are packed in a data message, such as a CAN data message. For input I/O modules, the network device may transmit CAN messages either periodically, on a change of state (% dead band), on level, or on a poll message. For output I/O modules, the network device may accept messages from the CAN bus to update the outputs.

In one embodiment, the network device allows an ability to prototype a device before hardware development. For example, the network device may be used as a CAN device instead of developing custom hardware.

In one embodiment, with the graphical configuration tool, the data messages, such as CAN data messages, can be packed with I/O data from different I/O modules, such data from the network interfaces 12-14 from the network module 10. In one embodiment, only input I/O data can only be packed into an input type data message and output I/O data type can only be packed into output type data message. As a result a data message, such as a CAN data message, can contain both analog and discrete data.

In one embodiment, this feature can be used to differentiate CAN message IDs from different physical sections of a device, such as a vehicle. This feature may be used to transmit analog data on a discrete trigger input, if both the analog channel and discrete channel are packed in one data message and the I/O and the data message are configured appropriately.

In one embodiment, the network module communicates using raw CAN network protocol, and may not need a master device to communicate with it, such as when using CAN Higher Level Protocols, or HLPs, such as CAL, CANOpen or DeviceNet, among others. The network device allows the user flexibility to configure any type of a data channel into a data message, such as a CAN data message. For example, the CAN device can be configured for either standard (11-bit) or extended (29-bit) message (arbitration) IDs.

Figures 12-14 – Graphical Configuration of I/O Points

Figure 12 illustrates configuration of an I/O point of the network device, according to one embodiment. Figure 13 illustrates configuration of a data message, according to one embodiment. Figure 14 illustrates saving of the configuration of the network device, according to one embodiment.

Data messages, such as CAN data messages, may be either of input or output type. Hence, an input type data message can contain I/O channels that can interface with sensors and an output type data message can contain I/O channel that can interface with actuators. If the end of the I/O interface is reached (all channels in the I/O interface are allocated), then the network device 10 may allocate another arbitration ID and continue with the next I/O interface. If an I/O interface consists of channels that can partially populate the entire 8 bytes of the data message, then some portion of the data message may be left unpopulated and a new arbitration ID may be allocated for the next I/O interface. If no more I/O interfaces are available, then the network device 10 may save the current configuration to the non-volatile memory.

The saved configuration may be retrieved and modified by accessing the network device via the host computer, such as via a configuration data message, running a configuration application and communicating over the network, such as the CAN bus. The configuration application, such as a graphical configuration tool, may query the hardware interface, such as described above with reference to Figures 9-11, for the allocated arbitration IDs and the I/O channels associated with the allocated arbitration IDs. In one embodiment, the configuration application may allow a user to override the information saved in the non-volatile memory, reconfigure, and save a new configuration. All of the allocated arbitration IDs may be deleted and new arbitration IDs may be allocated from the range of 253 arbitration IDs. At the time of allocation, the user may define if the data message, such as a CAN data message, is of input or output type. When an arbitration ID is deleted then all the I/O channels previously populating it may become unallocated. These channels may be allocated to a data message if the I/O type matches and if the space in the data message is adequate to accept the channel.

A configuration data message may be sent out that includes the information about the content of the data messages, including channel names, channel content, arbitration information, baud rate, and other configuration as described in Figures 7-14.

Figures 15-21 – Graphical Monitoring of Configured Channels and I/O Points

Figure 15 illustrates the graphical configuration tool accessing channels on the network, according to one embodiment. For example, the user can graphically access channels on the network by accessing the CAN Channels option in the Data Neighborhood folder using the graphical configuration tool.

Figure 16 illustrates the graphical configuration tool accessing channels on the network, according to one embodiment. For example, CAN channels with corresponding arbitration IDs may be displayed in the graphical configuration tool.

Figure 17 illustrates the user using the graphical configuration tool to monitor the network device, according to one embodiment. For example, the user can monitor channels and check properties for a network device.

Figure 18 illustrates the graphical configuration tool monitoring data on the network, according to one embodiment. For example, the data on an exemplary CAN interface 0 may be monitored.

Figure 19 illustrates accessing a test panel to test channels, according to one embodiment. For example, a test panel for an exemplary CAN Channel ID16777219 (0x1000003) may be accessed.

Figure 20 illustrates using the test panel to read and write values, according to one embodiment. For example, a test panel may be used to read and display values of an exemplary CAN Channel ID100001A_M1_C4 on a scope display. The values of the exemplary CAN channel may also be written.

Figure 21 illustrates accessing channel properties page, according to one embodiment. For example, CAN channel properties for an exemplary CAN Channel "AI Channel 1 on Demo Box" may be accessed to be viewed and/or changed.

Figures 22-23 – Using a Graphical Program to access Network Device

Figure 22 illustrates a diagram of a graphical program that accesses data on the network, according to one embodiment. Figure 23 illustrates a front panel of the graphical program that accesses data on the network, according to one embodiment.

As illustrated in Figure 22, a user may assemble a graphical program by selecting various icons or nodes, such as icons and nodes 51-55 as well as terminals 41A-45A, that represent desired functionality, and then connecting the nodes together to create the graphical program. The nodes or icons may be connected by lines representing data flow between the nodes, control flow, or execution flow. Thus the block diagram, such as the block diagram 50, may include a plurality of interconnected icons 51-55 such that the diagram created graphically displays a procedure or method for accomplishing a certain result, such as manipulating one or more input variables and/or producing one or more output variables. In response to the user constructing a diagram or a graphical program using the block diagram editor, data structures and/or program instructions may be automatically constructed which characterize an execution procedure that corresponds to the displayed procedure. The graphical program may be compiled or interpreted by a computer.

As illustrated in Figure 23, a graphical program may have a graphical user interface. For example, in creating a graphical program, a user may create a front panel 40 or a user interface panel. The front panel 40 may include various graphical user interface elements 41-45 or front panel objects, such as user interface controls and/or indicators, that represent or display the respective input and output that will be used by the graphical program, and may include other icons which represent devices being controlled. The graphical user interface elements 41-45 in the front panel may correspond to terminals 41A-45A in the block diagram respectively.

In one embodiment, a graphical program may communicate with one or more of the first network device and the second network device. The first data message is operable to be received and processed by the graphical program, such as the graphical program in Figures 22-23. The graphical program may be executed to perform an industrial

automation function, a process control function, and a test and measurement function, among others.

In another embodiment, an application program communicating with one or more of the first network device and the second network device includes a program created in one or more of a C, C++, Java, Visual Basic, and any other program development environment.

Although the embodiments above have been described in considerable detail, numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.